Terms used **compiler annotation non ambiguous memory optimize inline** Found **23,276** of **155,867**

Sort results by: relevance
Display results: expanded form
Save results to a Binder
Search Tips
Open results in a new window

Try an Advanced Search
Try this search in The ACM Guide

Results 1 - 20 of 200 Result page: **1** 2 3 4 5 6 7 8 9 10 next
Best 200 shown Relevance scale

1 MaJIC: compiling MATLAB for speed and responsiveness
George Almási, David Padua
May 2002 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation**, Volume 37 Issue 5
Full text available: pdf(619.32 KB) Additional Information: full citation, abstract, references, citings, index terms

This paper presents and evaluates techniques to improve the execution performance of MATLAB. Previous efforts concentrated on source to source translation and batch compilation; **MaJIC** provides an interactive frontend that looks like MATLAB and compiles/optimizes code behind the scenes in real time, employing a combination of just-in-time and speculative ahead-of-time compilation. Performance results show that the proper mixture of these two techniques can yield near-zero response time as ...

2 Using annotations to reduce dynamic optimization time
Chandra Krintz, Brad Calder
May 2001 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation**, Volume 36 Issue 5
Full text available: pdf(1.78 MB) Additional Information: full citation, abstract, references, citings, index terms

Dynamic compilation and optimization are widely used in heterogenous computing environments, in which an intermediate form of the code is compiled to native code during execution. An important trade off exists between the amount of time spent dynamically optimizing the program and the running time of the program. The time to perform dynamic optimizations can cause significant delays during execution and also prohibit performance gains that result from more complex optimization.

3 TIL: a type-directed optimizing compiler for ML
D. Tarditi, G. Morrisett, P. Cheng, C. Stone, R. Harper, P. Lee
May 1996 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1996 conference on Programming language design and implementation**, Volume 31 Issue 5
Full text available: pdf(1.23 MB) Additional Information: full citation, references, citings, index terms

4 Optimizing compilation of CLP(R)
Andrew D. Kelly, Kim Marriott, Andrew MacDonald, Peter J. Stuckey, Roland Yap
November 1998 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 20 Issue 6
Full text available: pdf(331.15 KB) Additional Information: full citation, abstract, references, citings, index terms, review

Constraint Logic Programming (CLP) languages extend logic programming by allowing the use of constraints from different domains such as real numbers or Boolean functions. They have proved to be ideal for expressing problems that require interactive mathematical modeling and complex combinatorial optimization problems. However, CLP languages have mainly been considered as research systems, useful for rapid prototyping, by not really

competitive with more conventional programming languages wh ...

Keywords: compilation, constraint logic programming, program analysis, program optimization, source-to-source program transformation

5 The benefits and costs of DyC's run-time optimizations

Brian Grant, Markus Mock, Matthai Philipose, Craig Chambers, Susan J. Eggers

September 2000 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 22 Issue 5

Full text available: pdf(1.59 MB) Additional Information: full citation, abstract, references, citings, index terms

DyC selectively dynamically compiles programs during their execution, utilizing the run-time-computed values of variables and data structures to apply optimizations that are based on partial evaluation. The dynamic optimizations are preplanned at static compile time in order to reduce their run-time cost; we call this staging. DyC's staged optimizations include (1) an advanced binding-time analysis that supports polyvariant specialization (enabling both single-way and multi ...

Keywords: dynamic compilation, specialization

6 The Jalapeño dynamic optimizing compiler for Java

Michael G. Burke, Jong-Deok Choi, Stephen Fink, David Grove, Michael Hind, Vivek Sarkar, Mauricio J. Serrano, V. C. Sreedhar, Harini Srinivasan, John Whaley

June 1999 **Proceedings of the ACM 1999 conference on Java Grande**

Full text available: pdf(1.34 MB) Additional Information: full citation, references, citings, index terms

7 C and tcc: a language and compiler for dynamic code generation

Massimiliano Poletto, Wilson C. Hsieh, Dawson R. Engler, M. Frans Kaashoek

March 1999 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 21 Issue 2

Full text available: pdf(471.68 KB) Additional Information: full citation, abstract, references, citings, index terms, review

Dynamic code generation allows programmers to use run-time information in order to achieve performance and expressiveness superior to those of static code. The 'C(Tick C) language is a superset of ANSI C that supports efficient and high-level use of dynamic code generation. 'C provides dynamic code generation at the level of C expressions and statements and supports the composition of dynamic code at run time. These features enable programmers to add dynamic code generation ...

Keywords: ANSI C, compilers, dynamic code generation, dynamic code optimization

8 Flick: a flexible, optimizing IDL compiler

Eric Eide, Kevin Frei, Bryan Ford, Jay Lepreau, Gary Lindstrom

May 1997 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1997 conference on Programming language design and implementation**, Volume 32 Issue 5

Full text available: pdf(1.75 MB) Additional Information: full citation, abstract, references, citings, index terms

An interface definition language (IDL) is a nontraditional language for describing interfaces between software components. IDL compilers generate "stubs" that provide separate communicating processes with the abstraction of local object invocation or procedure call. High-quality stub generation is essential for applications to benefit from component-based designs, whether the components reside on a single computer or on multiple networked hosts. Typical IDL compilers, ...

9 Compiler transformations for high-performance computing

David F. Bacon, Susan L. Graham, Oliver J. Sharp

December 1994 **ACM Computing Surveys (CSUR)**, Volume 26 Issue 4

Full text available: pdf(6.32 MB) Additional Information: full citation, abstract, references, citings, index terms, review

In the last three decades a large number of compiler transformations for optimizing programs have been implemented. Most optimizations for uniprocessors reduce the number

of instructions executed by the program using transformations based on the analysis of scalar quantities and data-flow techniques. In contrast, optimizations for high-performance superscalar, vector, and parallel processors maximize parallelism and memory locality with transformations that rely on tracking the properties o ...

Keywords: compilation, dependence analysis, locality, multiprocessors, optimization, parallelism, superscalar processors, vectorization

[10] Source-level global optimizations for fine-grain distributed shared memory systems

R. Veldema, R. F. H. Hofman, R. A. F. Bhoedjang, C. J. H. Jacobs, H. E. Bal

June 2001 **ACM SIGPLAN Notices , Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming**, Volume 36 Issue 7

Full text available: pdf(112.60 KB) Additional Information: full citation, abstract, references, citings, index terms

This paper describes and evaluates the use of aggressive static analysis in Jackal, a fine-grain Distributed Shared Memory (DSM) system for Java. Jackal uses an optimizing, source-level compiler rather than the binary rewriting techniques employed by most other fine-grain DSM systems. Source-level analysis makes existing access-check optimizations (e.g., access-check batching) more effective and enables two novel fine-grain DSM optimizations: object-graph aggregatio ...

[11] Developing an interprocedural optimizing compiler

Jon Loeliger

April 1994 **ACM SIGPLAN Notices**, Volume 29 Issue 4

Full text available: pdf(574.12 KB) Additional Information: full citation, abstract, citings, index terms

Compilers that perform interprocedural analysis require different user interfaces and different internal architectures than those used by compilers that process procedures independently. CONVEX Computer Corp. has developed a language-independent interprocedural optimizer that is packaged together with FORTRAN and C compilers in a product called the Application Compiler. User feedback and experience compiling large applications had a significant impact on the final user interface and system archi ...

[12] Memory optimization: A post-compiler approach to scratchpad mapping of code

Federico Angiolini, Francesco Menichelli, Alberto Ferrero, Luca Benini, Mauro Olivieri

September 2004 **Proceedings of the 2004 international conference on Compilers, architecture, and synthesis for embedded systems**

Full text available: pdf(154.30 KB) Additional Information: full citation, abstract, references, index terms

ScratchPad Memories (SPMs) are commonly used in embedded systems because they are more energy-efficient than caches and enable tighter application control on the memory hierarchy. Optimally mapping code and data to SPMs is, however, still a challenge. This paper proposes an optimal scratchpad mapping approach for code segments, which has the distinctive characteristic of working directly on application binaries, thus requiring no access to either the compiler or the application source code - a c ...

Keywords: design automation, dynamic programming, embedded design, executable patching, memory hierarchy, optimization algorithm, post-compiler processing, power saving, scratchpad memory

[13] Impact of economics on compiler optimization

Arch D. Robison

June 2001 **Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande**

Full text available: pdf(764.92 KB) Additional Information: full citation, abstract, references, citings, index terms

Compile-time program optimizations are similar to poetry: more are written than are actually published in commercial compilers. Hard economic reality is that many interesting optimizations have too narrow an audience to justify their cost in a general-purpose compiler, and custom compilers are too expensive to write. An alternative is to allow programmers to define their own compile-time optimizations. This has already happened accidentally for C++, albeit imperfectly, in the form of template ...

Keywords: compilers, economics, optimization

[14] An automatic object inlining optimization and its evaluation

Julian Dolby, Andrew Chien

May 2000 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation**, Volume 35 Issue 5

Full text available: pdf(877.17 KB) Additional Information: full citation, abstract, references, citings, index terms

Automatic object inlining [19, 20] transforms heap data structures by fusing parent and child objects together. It can improve runtime by reducing object allocation and pointer dereference costs. We report continuing work studying object inlining optimizations. In particular, we present a new semantic derivation of the correctness conditions for object inlining, and program analysis which extends our previous work. And we present an object inlining transformation, focusing ...

[15] The design and implementation of a certifying compiler

George C. Necula, Peter Lee

May 1998 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1998 conference on Programming language design and implementation**, Volume 33 Issue 5

Full text available: pdf(1.66 MB) Additional Information: full citation, abstract, references, citings, index terms

This paper presents the design and implementation of a compiler that translates programs written in a type-safe subset of the C programming language into highly optimized DEC Alpha assembly language programs, and a *certifier* that automatically checks the type safety and memory safety of any assembly language program produced by the compiler. The result of the certifier is either a formal proof of type safety or a counterexample pointing to a potential violation of the type system by the t ...

[16] Space and time-efficient memory layout for multiple inheritance

Peter F. Sweeney, Joseph (Yossi) Gil

October 1999 **ACM SIGPLAN Notices , Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications**, Volume 34 Issue 10

Full text available: pdf(2.30 MB) Additional Information: full citation, abstract, references, citings, index terms

Traditional implementations of multiple inheritance bring about not only an overhead in terms of run-time but also a significant increase in object space. For example, the number of compiler-generated fields in a certain object can be as large as quadratic in the number of its subobjects. The problem of efficient object layout is compounded by the need to support two different semantics of multiple inheritance: shared, in which a base class inherited along distinct ...

[17] Compiling nested data-parallel programs for shared-memory multiprocessors

Siddhartha Chatterjee

July 1993 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 15 Issue 3

Full text available: pdf(4.17 MB) Additional Information: full citation, references, citings, index terms, review

Keywords: compilers, data parallelism, shared-memory multiprocessors

[18] New garbage collection algorithms and strategies: Dynamic selection of application-specific garbage collectors

Sunil Soman, Chandra Krintz, David F. Bacon

October 2004 **Proceedings of the 4th international symposium on Memory management**

Full text available: pdf(185.74 KB) Additional Information: full citation, abstract, references, index terms

Much prior work has shown that the performance enabled by garbage collection (GC) systems is highly dependent upon the behavior of the application as well as on the available resources. That is, no single GC enables the best performance for all programs and all heap sizes. To address this limitation, we present the design, implementation, and empirical evaluation of a novel Java Virtual Machine (JVM) extension that facilitates dynamic switching between a number of very different and popular g ...

Keywords: Java, annotation, application-specific collection, dynamic selection, hot-swapping, virtual machine

[19] Scalable cross-module optimization

Andrew Ayers, Stuart de Jong, John Peyton, Richard Schooler

May 1998 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1998 conference on Programming language design and implementation**, Volume 33 Issue 5

Full text available: pdf(1.48 MB) Additional Information: full citation, abstract, references, citings, index terms

Large applications are typically partitioned into separately compiled modules. Large performance gains in these applications are available by optimizing across module boundaries. One barrier to applying crossmodule optimization (CMO) to large applications is the potentially enormous amount of time and space consumed by the optimization process.We describe a framework for scalable CMO that provides large gains in performance on applications that contain millions of lines of code. Two major techni ...

[20] Techniques for obtaining high performance in Java programs

Iffat H. Kazi, Howard H. Chen, Berdenia Stanley, David J. Lilja

September 2000 **ACM Computing Surveys (CSUR)**, Volume 32 Issue 3

Full text available: pdf(816.13 KB) Additional Information: full citation, abstract, references, citings, index terms

This survey describes research directions in techniques to improve the performance of programs written in the Java programming language. The standard technique for Java execution is interpretation, which provides for extensive portability of programs. A Java interpreter dynamically executes Java bytecodes, which comprise the instruction set of the Java Virtual Machine (JVM). Execution time performance of Java programs can be improved through compilation, possibly at the expense of portabili ...

Keywords: Java, Java virtual machine, bytecode-to-source translators, direct compilers, dynamic compilation, interpreters, just-in-time compilers

Results 1 - 20 of 200 Result page: **1** 2 3 4 5 6 7 8 9 10 next